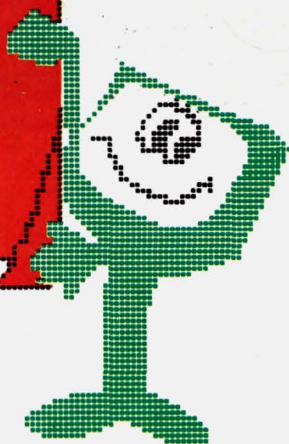
20 VIDEOLEZIONI DI BASIC
PER IMPARARE COL VIC 20

GRUPPO
EDITORIALE
JACKSON

I computer del futuro
L'intelligenza artificiale

Il BASIC e la memorizzazione
dei programmi

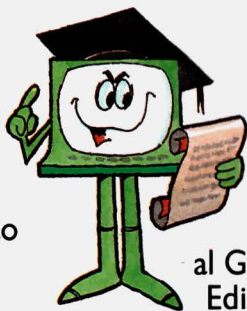
Gli interrupt

Usare la ROM

Videoesercizi

Videogioco n° 20

20

COMMODORE
VIC 20

VIDEO BASIC VIC 20
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
Direttore Responsabile:
Giampietro Zanga
Direttore e Coordinatore
Editoriale: Roberto Pancaldi
Autore: Softidea - Via Indipendenza 88 - Como
Redazione software:
Francesco Franceschini, Enrico Braglia,
Fabio Calanca
Segretaria di Redazione:
Marta Menegardo
Progetto grafico:
Studio Nuovaidea - Via Longhi 16 - Milano
Impaginazione:
Silvana Corbelli
Illustrazioni:
Cinzia Ferrari, Silvano Scolari
Fotografie:
Marcello Longhini
Distribuzione: SODIP
Via Zuretti, 12 - Milano
Fotocomposizione: Lineacomp S.r.l.
Via Rosellini, 12 - Milano
Stampa: Grafika '78
Via Trieste, 20 - Pioltello (MI)
Direzione e Redazione:
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5

Autorizzazione alla pubblicazione Tribunale di
Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70
(autorizzazione della Direzione Provinciale delle
PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo
Editoriale Jackson S.r.l. - Via Rosellini, 12
20124 Milano, mediante emissione di assegno
bancario o cartolina vaglia oppure
utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere
richiesti direttamente all'editore
inviando L. 10.000 cdu. mediante assegno
bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

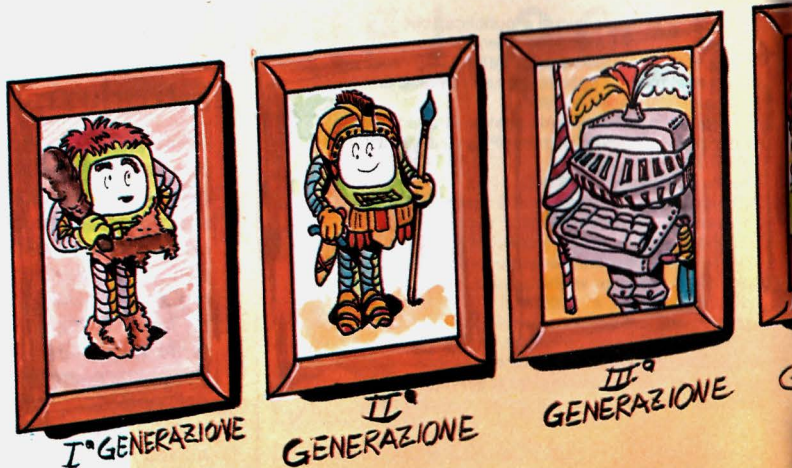
Gruppo Editoriale Jackson

SOMMARIO

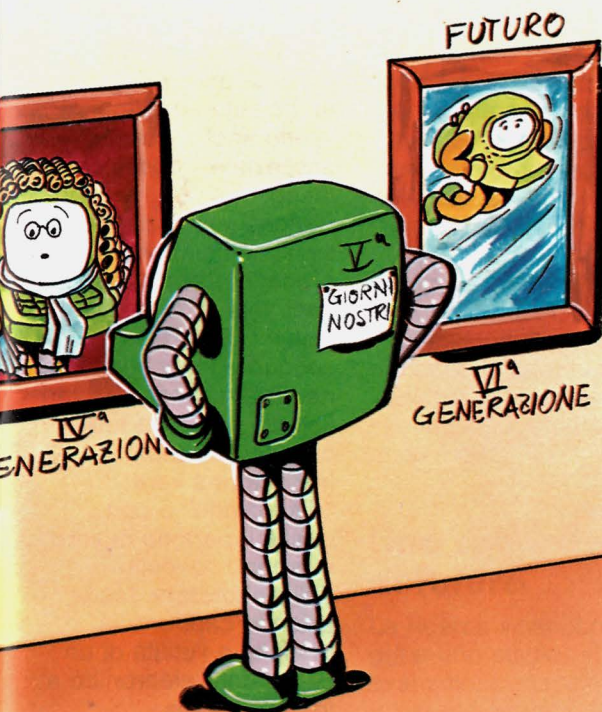
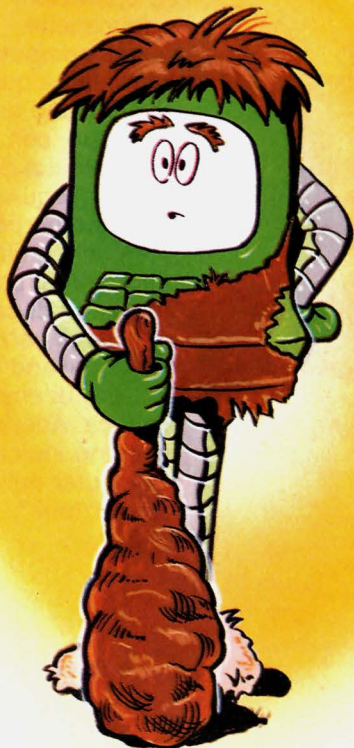
Introduzione

Il tuo VIC 20 ha qualità e pregi impensabili sino a pochi anni fa; quello che i laboratori di ricerca ci stanno preparando, però, va ben oltre: computer ultraveloci, capaci di elaborare più informazioni in parallelo, ma soprattutto computer "intelligenti". La fantascienza, insomma, non è più tanto futuribile.
Per restare, comunque, coi piedi nel presente dobbiamo perfezionare quanto appreso e apprendere del nuovo.
Ecco allora come risparmiare tempo e memoria, i puntatori, gli interrupt, e ancora linguaggio macchina.
Morale. Se il computer non ragiona e parla ancora come un uomo, occorre saper ragionare e parlare come il computer.

I computer del futuro

Per quanto molte persone ritengano tuttora il contrario, i computer non lavorano assolutamente per magia: ormai sappiamo benissimo che tutti i risultati ottenibili mediante un elaboratore elettronico non sono altro che il prodotto di un complesso e velocissimo insieme di operazioni elementari, svolte all'interno dei circuiti e delle memorie della macchina. Ciò che ad alcuni può ancora sembrare fantastico, è in realtà un preciso e logico sviluppo di un settore tecnologico che poggia su solide e rigorose basi teoriche e scientifiche.
Soltanto fino a pochi decenni fa nessuno avrebbe tuttavia potuto pronosticare la nascita e soprattutto lo sviluppo di una tecnologia così rivoluzionaria come quella elettronica: non esisteva infatti alcun presupposto che lasciasse intravvedere un avvenire così carico di sviluppi. Adesso che viviamo in pieno nella cosiddetta "era

elettronica" abbiamo invece sufficienti "orizzonti" per poter immaginare quali saranno i probabili progressi nel settore dei computer.
Le strade che in questo momento vengono battute dagli studiosi di tutto il mondo meritano comunque di essere descritte e analizzate.

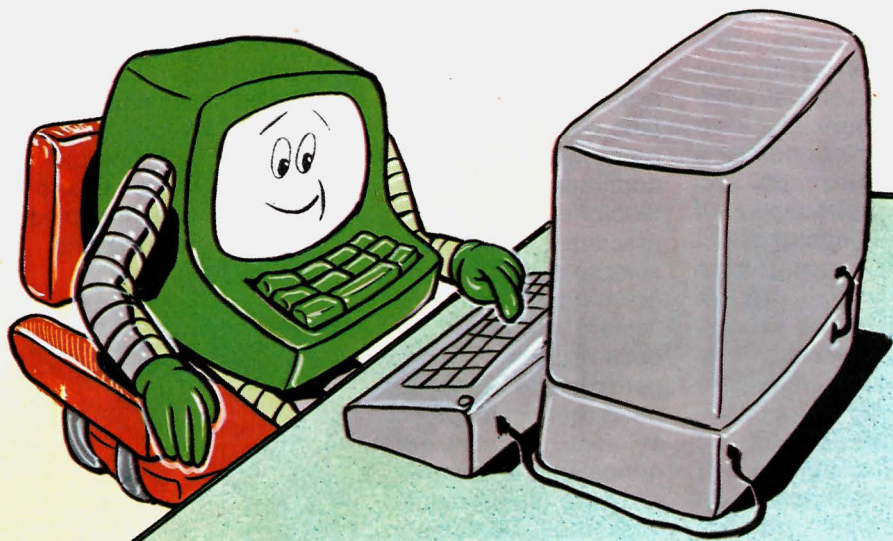
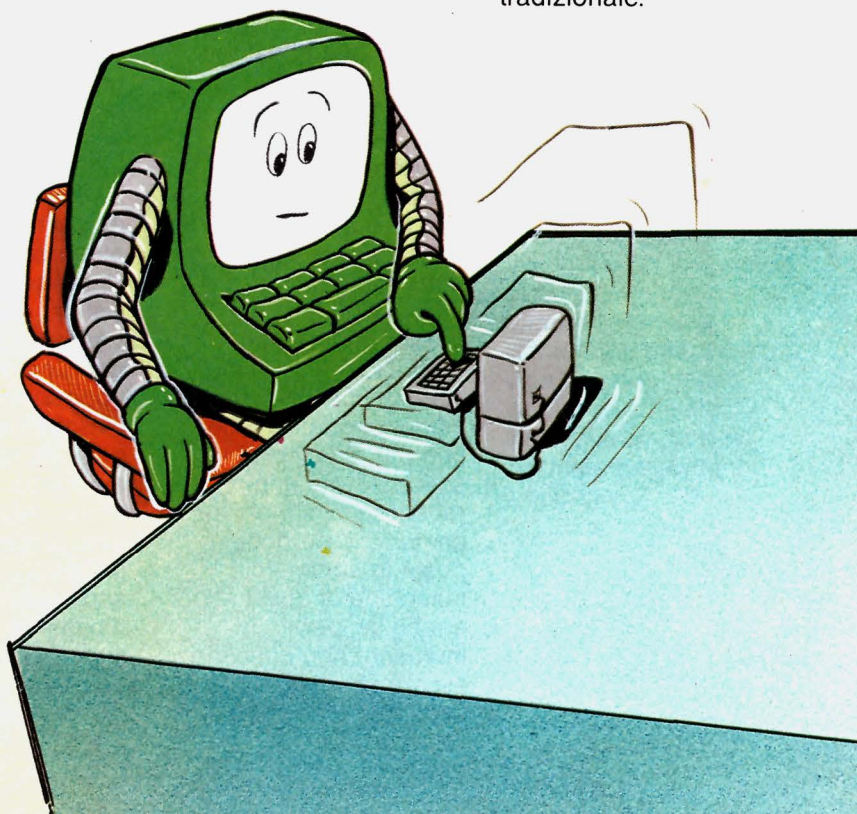
I calcolatori del passato

Prima di pensare al futuro diamo innanzitutto un'occhiata al passato: il detto "preparati al futuro guardando nel passato" è più che mai appropriato per chi, come noi, desidera fondare in maniera concreta le proprie ipotesi. Trascurando gli studi e le teorie logiche che hanno consentito la nascita e l'evoluzione del calcolo automatico, andiamo all'inizio degli anni '50, quando ancora nel campo elettronico imperavano le valvole e i diodi. La guerra era finita da poco e i calcolatori erano appena agli albori: in quel periodo cominciarono comunque ad apparire le prime macchine elettroniche capaci di eseguire automaticamente calcoli ed operazioni matematiche.
Questi computer raffrontati a quelli di oggi sembravano dei dinosauri, con dimensioni a dir poco enormi (l'equivalente di un moderno personal occupava un intero laboratorio), composti da chilometri di cavi e migliaia di valvole. In più consumavano notevoli quantità di energia elettrica. Adesso vengono indicati come "computer della prima generazione".

Le macchine di questo tipo erano principalmente a disposizione dei grossi centri di ricerca militari od universitari e di grandi industrie: l'hardware era infatti troppo ingombrante e soggetto a guasti per garantire immediati sviluppi o applicazioni commerciali. Anche per il software le cose non andavano molto meglio: linguaggio macchina e linguaggi simbolici elementari di tipo assemblatore ponevano grossi limiti di utilizzo.
Però, pian piano gli elaboratori cominciarono comunque a diffondersi: occorreva però renderli più affidabili e meno ingombranti. Il fatto decisivo fu la scoperta del transistor. In un computer un transistor assolve la stessa funzione di una valvola: il transistor consuma però molto meno energia delle valvole, è di dimensioni molto minori e soprattutto dispone di una vita media di gran lunga superiore. Inoltre, - fatto anche questo essenziale - costano meno.
Benché sostanzialmente simili agli elaboratori precedenti sotto l'aspetto della logica, questi nuovi sistemi (detti della seconda generazione) se ne distaccavano per vari aspetti, primo tra tutti il dimensionamento.
Iniziava una vera "rivoluzione" e con quella il periodo di vero e proprio "decollo" dell'elaboratore. Molte aziende capirono la praticità e l'utilità di un calcolatore elettronico e lo installarono

richiedendo nello stesso tempo una maggiore potenza e ancora più elevata velocità di elaborazione. Nuove possibilità di elaborazione apparvero con l'introduzione delle memorie a nucleo magnetico, allargando ulteriormente le applicazioni e i possibili sviluppi.
Il più grosso passo avanti venne comunque fatto con la creazione dei circuiti integrati o chip: questa nuova tecnologia generò gli elaboratori della "terza generazione". I circuiti integrati introdussero nuovi miglioramenti, miniaturizzando e raffinando i componenti della seconda generazione. Essi inoltre costavano ancora meno delle "vecchie" piastre a transistor.
Contemporaneamente agli sviluppi dell'hardware, anche la programmazione aveva fatto passi da gigante, proponendo in continuazione nuove tecniche e applicazioni, grazie a linguaggi sempre più potenti e nello stesso tempo più "umani". I computer della "quarta generazione" sono quelli dei giorni nostri: piccoli, efficienti, affidabili, ma - nonostante le apparenze - ancora ulteriormente migliorabili. Vediamo in quali modi.

Uno sguardo al futuro

La ricerca scientifica è già a uno stadio avanzato: si stanno infatti battendo molte strade, sia "hardware" che "software". Le principali aree di studio dal punto di vista costruttivo riguardano soprattutto la superconduttività e l'elaborazione parallela, mentre l'informatica vera e propria punta tutte le proprie speranze verso quel settore di indagine, estremamente stimolante, che prende il nome di "intelligenza artificiale". Al solito, lo scopo è quello di realizzare elaboratori sempre migliori, sempre più versatili, sempre più utili; che lavorino più in fretta, memorizzino più informazioni, richiedano meno potenza, occupino meno spazio e costino sempre meno.

I computer a superconduttori

In realtà, i computer a superconduttori esistono già da qualche tempo: tuttavia le possibilità di sviluppo che essi sembrano in grado di offrire li pongono in un settore che appartiene più al domani che all'oggi.
I computer attualmente costruiti hanno raggiunto velocità di elaborazione talmente elevate da poter essere confrontate alle velocità con cui gli elettroni si muovono nei circuiti elettronici. È chiaro che se gli spostamenti degli elettroni all'interno dell'elaboratore (ricordiamoci che il flusso degli elettroni all'interno dei circuiti costituisce la corrente elettrica) sono più lenti delle possibilità di calcolo dell'unità centrale, il tempo di esecuzione è costretto a subire un rallentamento.
In altre parole, le informazioni impiegano un certo tempo (per quanto piccolissimo) per

andare da un punto all'altro dei circuiti elettronici: se questo tempo è superiore a quello della velocità di calcolo della CPU, questa è per forza di cose costretta a "rallentare", con ovvie conseguenze sulla velocità di lavoro dell'intero sistema. Questo problema, a prima vista apparentemente insolubile (non esiste infatti alcuna possibilità di "accelerare" il moto degli elettroni nei conduttori, essendo quest'ultimo una caratteristica specifica dei materiali), viene allora risolto ricorrendo a particolari leghe di metalli conduttori, che godono della singolare proprietà di opporre - a temperature estremamente basse (più di 100 gradi al di sotto dello zero!) - una resistenza al moto degli elettroni di gran lunga più bassa di quella che offrono alle normali temperature ambientali. Il problema viene quindi brillantemente risolto: le velocità di elaborazione dei computer a superconduttori (superconduttività è il nome del processo fisico che abbiamo appena visto) raggiungono infatti valori assolutamente impensabili nei comuni computer a tecnologia tradizionale.

I computer paralleli

L'idea che sta alla base della tecnica dei computer paralleli è di una semplicità quasi disarmante: anziché usare una sola unità centrale, nei computer paralleli si utilizzano più unità centrali, che lavorano in modo simultaneo o, come si *usa dire più* comunemente, in parallelo. Le possibilità che si propongono sono estremamente allettanti: teoricamente basta aggiungere altre CPU e le potenzialità di elaborazione di qualsiasi computer diventano praticamente senza limiti.

Naturalmente, come nella maggior parte dei progetti che a parole sembrano "semplici", gli spazi di azione che separano il dire dal fare appaiono estremamente impegnativi.
I problemi principali dei computer paralleli non risiedono infatti unicamente nelle pure e semplici disposizioni circuitali (tutt'altro che facili da risolvere), ma anche (e soprattutto) nelle modalità di programmazione che queste macchine richiedono.
Occorre infatti disporre di un linguaggio che riesca a "sincronizzare" le operazioni di tutte le CPU, evitando interferenze ed accavallamenti reciproci, con le ovvie (e deleterie) conseguenze che ne potrebbero derivare. A tutt'oggi un linguaggio di questo genere non esiste: i tentativi finora sperimentati lasciano comunque intravvedere ben più di semplici speranze. In tempi recentissimi sono state comunque prodotte e poste in commercio macchine di questo tipo, anche se, per il momento non possono essere sfruttate al pieno delle loro possibilità.

Intelligenza artificiale

L'intelligenza artificiale è senza alcun dubbio il settore che nello sviluppo della scienza dei computer avrà la maggiore influenza sul nostro modo di vivere nei prossimi decenni.
Per generazioni gli scrittori di fantascienza hanno pronosticato l'evoluzione di macchine più o meno intelligenti, capaci di assolvere molte delle funzioni eseguibili soltanto dagli esseri umani. Con ogni probabilità le vicende di androidi e umanoidi resteranno però di esclusivo dominio della lettura fantastica e avveniristica.
Al giorno d'oggi si pensa piuttosto alle macchine "intelligenti" come a sistemi capaci di prendere determinate decisioni al momento più opportuno.
L'esatta definizione dell'intelligenza di una macchina è un argomento in continua evoluzione: per quanto gli esperti si accaniscano in continui dibattiti su questo tema, si può comunque accettare come definizione standard

quella che venne proposta nei "lontani" anni '40 da un vero e proprio pioniere dell'informatica: Alan Turing. Piuttosto che elencare una serie di criteri da soddisfare per classificare un computer come intelligente, egli si limitò a dare una opinione ben più pratica del problema.
Turing affermò che se una persona non è in grado di distinguere se certe risposte le arrivano da una macchina o da un'altra persona, allora la macchina che ha eventualmente elaborato quelle risposte è da classificarsi come intelligente. Tale prova costituisce la base del famoso "Test di Turing", nel quale un operatore umano deve sostenere - attraverso una tastiera e un terminale - una certa conversazione, cercando di costringere l'interlocutore a svelare la propria identità di uomo o di macchina.
I centri di ricerca di tutto il mondo stanno portando avanti studi e indagini sulla intelligenza artificiale, e sembra che anche in questo caso sia solo questione di anni per arrivare a risultati effettivi. Il Giappone ha già anticipato tutti, annunciando che il suo elaboratore della "quinta generazione" vedrà la luce al massimo entro il 1995.
L'intelligenza artificiale ha già fatto il suo ingresso in molti settori, come per esempio quello dei "sistemi esperti": con questo nome si intendono quegli elaboratori specializzati in grado di eseguire un certo compito altrettanto bene (e in alcuni casi anche meglio) degli esperti umani. Un tipico esempio di utilizzo di questa tecnologia lo possiamo vedere tutti i giorni guardando le previsioni del tempo, elaborate quotidianamente appunto mediante l'ausilio di sistemi esperti. Anche nel campo delle diagnosi mediche sembra si stiano facendo passi da gigante: l'ipotesi del computer-dottore non è quindi troppo azzardata.
La maggior parte degli studi sull'intelligenza artificiale vengono condotti utilizzando particolari linguaggi, creati appositamente per questo scopo (in genere LISP e PROLOG). Poiché questi ultimi richiedono potenze di calcolo ben superiori a quelle offerte dai piccoli computer, nelle nostre case l'intelligenza artificiale entrerà tra pochi anni grazie alla telematica che consentirà di collegare l'home computer ad un grande sistema intelligente.
Ciò non significa comunque che il campo non sia affrontabile - anche se a livello hobbystico - con un normale personal computer.

Risparmiare tempo e memoria

Nella programmazione, come in molti altri settori del lavoro umano, è stata costruita una scala di valori, che classifica le varie tecniche in "buono", "non molto buono", "pessimo". Quando hai cominciato a usare il tuo VIC 20, essere in grado di scrivere un programma funzionante era già di per sé un valido risultato. Adesso che sei diventato molto più padrone della situazione, e disponi di conoscenze e capacità che inizialmente non avevi, è giunto il momento di affrontare un discorso che - a prima vista - ti potrebbe sembrare poco rilevante, ma che in realtà è importantissimo per migliorare la tua tecnica di programmazione.
Vogliamo infatti vedere come migliorare ed aumentare la velocità di esecuzione dei tuoi programmi BASIC, senza per questo influenzarne la qualità, la leggibilità e l'occupazione di memoria. Non sempre velocità di esecuzione e risparmio di memoria sono conciliabili: esiste comunque un certo numero di "trucchetti", che di volta in volta possono tornare utili. Vediamone quindi alcuni:

- **Istruzioni multiple.**
Il porre più istruzioni in una stessa linea aiuta a minimizzare la lunghezza del programma, risparmiando sui numeri di linea e di conseguenza sull'occupazione di memoria. Lo svantaggio e la limitazione principale di questa tecnica risiede però nella scarsa leggibilità e nella difficile modificabilità delle varie righe. È quindi meglio non abusare troppo con questa pratica.

- **Variabili.**
Le variabili dovrebbero essere chiamate con i nomi più brevi possibili. Questo aiuta a risparmiare memoria ed accelera il lavoro dell'interprete BASIC. Anche le costanti (sia numeriche che alfanumeriche) - se utilizzate di frequente - conviene assegnarle a delle variabili. Per esempio, anziché scrivere:

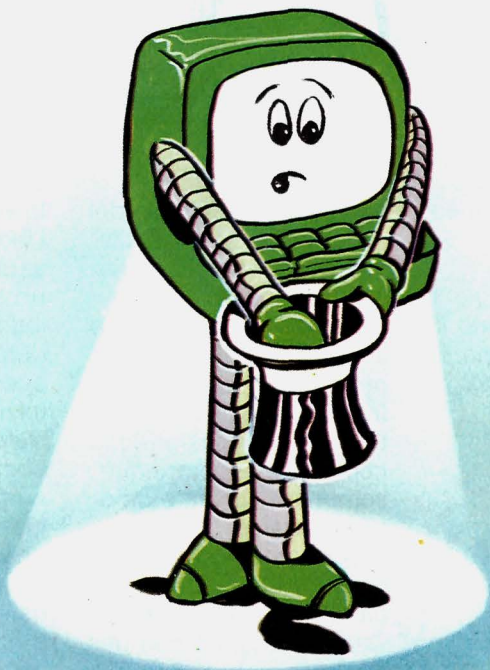
```
10 POKE 15143,12:POKE 15143,70:POKE 15143,20
```

conviene fare:

```
10 LET A=15143:POKE A,12:POKE A,70:POKE A,20
```

L'interprete BASIC dovrà in questo modo convertire una sola volta il numero 15143 in un formato comprensibile alla CPU, riducendo oltretutto anche lo spazio occupato in memoria.

• REM.

Bisogna limitare al massimo l'uso dei commenti nei programmi. Questa indicazione sembra in apparente contraddizione con quanto avevamo detto finora, e cioè che la documentazione dei programmi dovrebbe essere la più abbondante possibile. I programmatori più esperti risolvono allora il problema conservando due copie dello stesso programma: la prima, commentata in tutti i suoi aspetti, serve per capire il funzionamento del programma e per apportarvi eventuali modifiche; la seconda, che dovrà girare nel computer, viene privata di qualsiasi commento, in modo da evitare "inutili" sprechi di memoria.

● **GOTO.**
Ogni volta che l'interprete deve eseguire un salto la sequenza delle istruzioni subisce un improvviso cambiamento, spostandosi a un altro punto del programma. L'entità di questo cambiamento viene definita specificando nell'istruzione GOTO il numero di linea a cui saltare. L'interprete BASIC non dispone però di tecniche di ricerca particolari per individuare tale numero di linea ed esegue quindi una lenta ricerca sequenziale a partire dall'inizio del programma. Il tempo necessario per eseguire una istruzione di salto dipende dunque dalla lunghezza del testo e dalla distanza della linea indicata nel GOTO dall'inizio del testo. Quando si desidera accelerare al massimo la velocità di esecuzione è pertanto necessario valutare attentamente questo fatto, cercando di limitare al massimo la presenza di GOTO (avvantaggiando anche la leggibilità del programma) o - se proprio non se ne può fare a meno - avvicinando al massimo le istruzioni a cui saltare all'inizio del programma.

● **GOSUB.**
L'utilizzo delle subroutine consente un notevole risparmio di memoria, dal momento che permette di evitare la ripetuta scrittura di gruppi di istruzioni. Per i comandi GOSUB valgono tuttavia le stesse considerazioni fatte per i GOTO; la cosa migliore (e questo è di semplice applicazione) è allora quella di porre tutte le subroutine - all'inizio dei programmi, anziché alla fine come siamo abituati - dando la precedenza (cioè scrivendole prima) a quelle che il programma utilizzerà più frequentemente.
La raccomandazione più importante è comunque sempre la stessa e cioè che qualsiasi programma può diventare notevolmente più corto e veloce, se viene strutturato con una buona logica.

Come lavora il BASIC: i puntatori

Un puntatore è una porzione della memoria del computer (di solito molto piccola: una o due locazioni) il cui contenuto è costituito da indirizzi utili al funzionamento del sistema. Ci spieghiamo meglio con un esempio. Quando accendi il tuo VIC 20, l'interprete BASIC deve mettersi immediatamente nella condizione di accettare le linee di programma che vorresti battere. Per fare questo è chiaramente necessario che l'interprete conosca la zona di memoria in cui dovrà immagazzinare le varie istruzioni; l'indirizzo di tale zona sarà allora contenuto in un apposito puntatore, letto dall'interprete durante la routine di accensione (o inizializzazione) del sistema. In ogni computer esistono numerosi puntatori, ciascuno dei quali con una specifica funzione. La loro principale utilità risiede nel fatto che grazie ad essi è possibile riferirsi con

minimo sforzo a particolari aree della memoria, consentendo quindi, con estrema facilità, eventuali modifiche o aggiornamenti. Tra breve scopriremo che anche la memorizzazione delle linee di programma ricorre ampiamente a questa tecnica.

La memorizzazione dei programmi

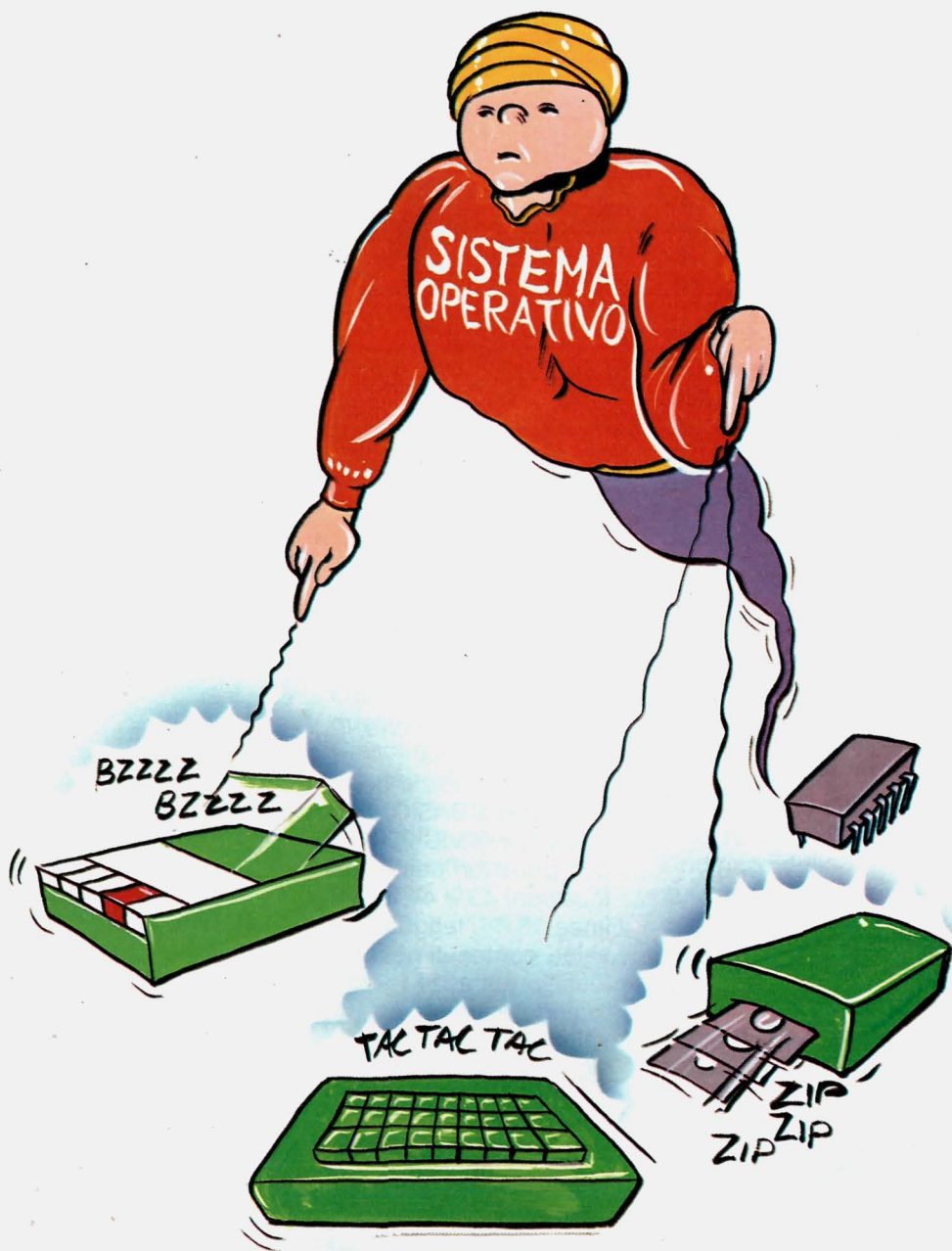
Quando batti in BASIC una qualsiasi riga di programma le linee vengono normalmente memorizzate (escludendo eventuali "modifiche" ai puntatori) a partire dal byte

4096

puntato dai due byte 43 e 44 della pagina zero.
La struttura delle istruzioni è la seguente:

- 2 byte di LINK (collegamento), che contengono l'indirizzo della successiva linea di programma; l'ultima istruzione del programma - non avendo alcuna riga dopo - ha i due byte di LINK entrambi a zero. Come sempre accade per i byte utilizzati a coppie, il primo byte è quello meno significativo, e il secondo il più significativo.
- 2 byte contenenti il numero della linea BASIC: prima il meno significativo, poi quello più significativo.
- Il testo delle parole BASIC scritte in codice ASCII, dove:
  — le parole e i simboli riservati occupano un solo byte. Per risparmiare memoria le parole riservate vengono infatti convertite in particolari codici numerici, chiamati TOKEN. Così, per esempio, la parola GOTO non viene memorizzata con 4 lettere separate ("G", "O", "T", "O") e quindi con 4 byte, ma con l'unico codice 137;
  — tutte le altre parti che compongono la linea sono invece espresse carattere per carattere.
- La linea termina con un byte a zero.

Da quanto abbiamo appena detto risulta che tre byte consecutivi a zero segnalano la fine del programma (i tre byte sono rispettivamente il byte terminale dell'ultima linea e i due byte di LINK dell'ipotetica linea successiva).
Se la linea BASIC contiene degli spazi tra una parola e l'altra, essi vengono mantenuti con "spreco" della memoria, ma con guadagno della leggibilità. Non viene invece conservato lo spazio tra il numero di linea e il primo carattere della riga; questo spazio viene tuttavia aggiunto in fase di listing del programma (cioè quando desideri leggere ciò che hai battuto).
Segue un programma che illustra quanto

SISTEMA OPERATIVO
BZZZZ
BZZZZ
TAC TAC TAC
ZIP
ZIP
ZIP

appena visto. Esso infatti "legge" se stesso nella memoria del VIC 20.

```
 1 REM PROVAMEN
 5 OPEN4,4:CMD4:PRINT"PROGRAMMA
   IN MEMORIA":PRINT
10 X=256*PEEK(44)+PEEK(43)
15 Y1=PEEK(X):Y2=PEEK(X+1)
17 IF Y1+Y2=0 THEN PRINT X;"**";Y1;Y2:GOTO60
20 PRINT X;"**";Y1;Y2;"LINK=";Y2*256+Y1
25 Y1=PEEK(X+2):Y2=PEEK(X+3)
30 PRINT "NUMERO LINEA:  ";Y2*256+Y1
35 X=X+4:C=0
37 Y1=PEEK(X):IF Y1=0 THEN 55
40 PRINT Y1;"   ";:X=X+1:C=C+1
45 IF C=6 THEN PRINT:C=0
50 GOTO 37
55 PRINT Y1:X=X+1:GOTO 15
60 PRINT#4:CLOSE4:STOP
```

Vediamo brevemente il significato delle varie istruzioni:
linea 5: apre il canale della stampante e stampa l'intestazione;
linea 10: calcola l'indirizzo di inizio del programma BASIC, indicato nel VIC 20 dai due puntatori posti nelle locazioni 43 e 44;
linee 15-17: leggono i valori contenuti nelle locazioni X e X+1: quando entrambi questi valori saranno nulli, significherà che siamo arrivati agli ultimi due byte di LINK e il programma dovrà terminare;
linee 20-55: costituiscono il ciclo principale del programma e stampano il contenuto di tutti i byte che nella memoria del VIC 20 compongono il programma;
linea 60: chiude il canale della stampante e arresta il programma.

Eseguendo il programma otterrai in uscita una cosa del genere:

```
PROGRAMMA IN MEMORIA

4096 ** 15 16 LINK= 4111
NUMERO LINEA: 1
143                32  80  82  79  86
65                 77  69  77  0
4111 ** 53 16 LINK= 4149
NUMERO LINEA: 5
159                52  44  52  58  157
52                 58  153 34  80  82
79                 71  82  65  77  77
65                 32  73  78  32  77
69                 77  79  82  73  65
34                 58  153 0
......
```

e così via.

Cerchiamo adesso di capire cosa significano queste serie di numeri. Ciascun gruppo di valori è la rappresentazione numerica con cui il tuo VIC 20 ha memorizzato le istruzioni. Per esempio, in corrispondenza di

```
1 REM PROVAMEN
```

troviamo:
4096 che è l'indirizzo da cui parte la memorizzazione dell'istruzione.

15 16: sono i due byte di link (infatti 16*256+15=4111 è l'indirizzo dell'istruzione successiva)

1 è il numero della linea, ottenuto leggendo il terzo e il quarto byte del gruppo di locazioni appartenenti alla linea stessa (vedi righe 25-30 del programma BASIC)

143 è il token della parola riservata REM

32 è il codice ASCII del carattere spazio

80 82 79 86 65 77 69 77 sono i codici ASCII dei caratteri della parola "PROVAMEN"

0 è l'indicatore di fine linea

4111 è l'indirizzo da cui parte la memorizzazione della nuova linea di programma (come sapevamo già dal valore del precedente link)

53 16: sono i due byte di link (16*256+53=4149 è l'indirizzo della successiva istruzione)

5 è il numero della linea, ottenuto nel modo visto in precedenza

159 è il token della parola riservata OPEN

52 44 52 58 sono i codici ASCII dei caratteri 4, 4:

157 è il token di CMD

52 58 sono i codici rispettivamente di 4 e :

153 è il token di PRINT

34 80 82....65 3458 sono i codici ASCII di "PROGRAMMA IN MEMORIA":

153 è sempre il token di PRINT

0 è l'indicatore di fine linea.

Le altre linee del programma possono essere analizzate nel modo appena visto: come esercizio lo lasciamo fare a te.
Ti proponiamo invece un altro interessante programma, il cui compito è quello di stampare tutte le parole riservate che in BASIC vengono "tokenizzate", cioè ridotte a un solo codice numerico. Tali codici coprono l'intervallo dei numeri da 128 a 202, ma non possono mai essere scambiati con i normali caratteri corrispondenti a questi numeri, visto che vengono interpretati come parole chiave nel contesto nel quale sono presi in esame.

```
 1 REM TOKEN
 5 OPEN 4,4:CMD4
 9 L$="":J=128
10 FOR K=49310 TO 49567
15 L=PEEK(K)
20 IF L<=127 THEN L$=L$+CHR$(L):NEXT K
25 L=L-128:L$=L$+CHR$(L)
30 PRINT J, L$
40 J=J+1:NEXT K
50 PRINT#4:CLOSE 4: STOP
```

Il programma, dopo aver aperto la stampante alla linea 5, inizializza le due variabili L$ e J che conterranno rispettivamente le parole chiave e i codici decimali delle stesse.
Alla linea 10 inizia un ciclo FOR,con K che varia da

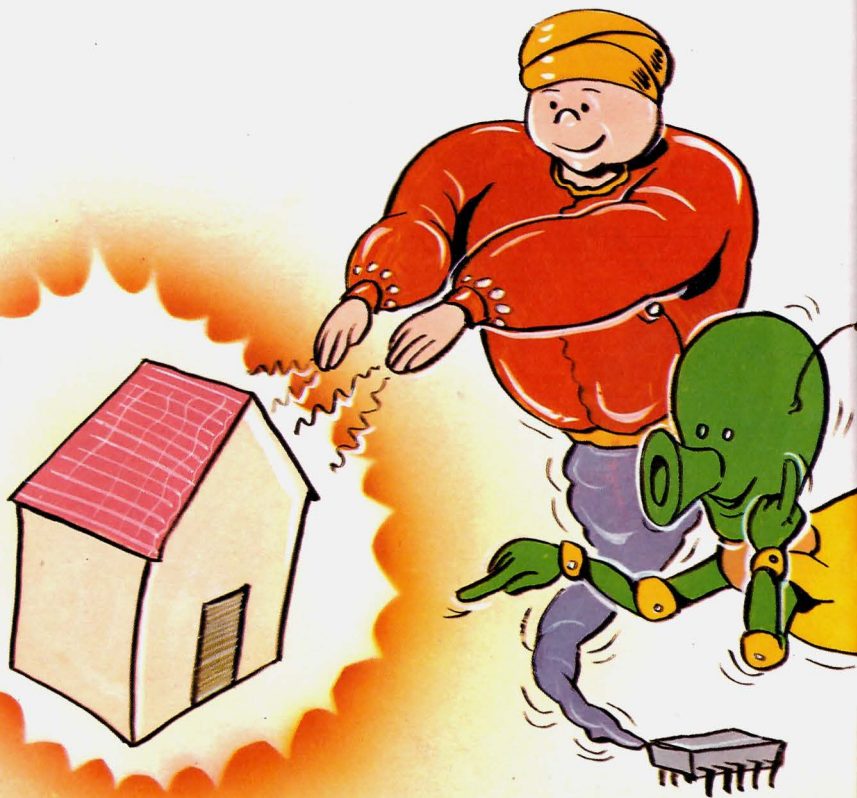
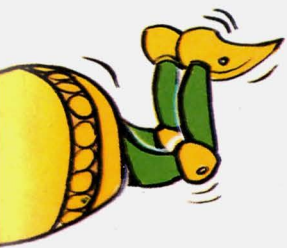
49310 a 49567

Questi due numeri sono gli indirizzi che delimitano una zona di memoria ROM dove sono memorizzate le parole chiave.
La tecnica di memorizzazione è la seguente: una parola chiave viene memorizzata carattere per carattere con i

LINGUAGGIO

codici ASCII corrispondenti; per segnalarne la fine viene sommato 128 al codice dell'ultimo carattere. Nel programma si analizzano i codici letti, e quando un codice supera 127, si considera conclusa la parola, procedendo di conseguenza alla stampa del token e della relativa parola riservata. Una volta concluso il ciclo la linea 50 chiude il canale aperto in precedenza e arresta il programma.

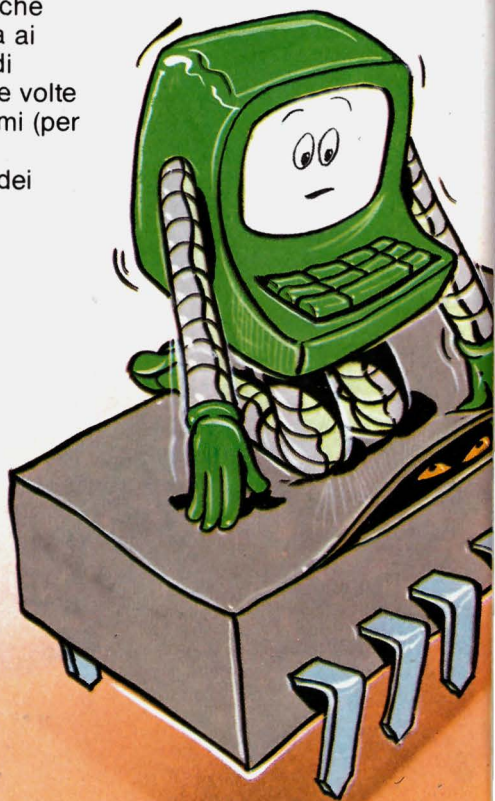
Gli interrupt

La CPU usa l'area di stack per numerose operazioni, tra le quali anche la gestione degli interrupt.
La comunicazione tra la CPU e le periferiche può avvenire con due tecniche diverse. La prima prende il nome di

"polling" (interrogazione ciclica): la CPU interroga ciclicamente, secondo un ordine prestabilito, i dispositivi esterni, usando un programma che legge lo stato degli stessi. Quando uno dei dispositivi è pronto a trasmettere o a ricevere un dato la CPU manda il programma necessario all'operazione richiesta.
La priorità tra i diversi dispositivi è determinata dall'ordine con cui questi ultimi vengono interrogati.
Questa tecnica presenta un solo vantaggio, e cioè di essere realizzata completamente via software, e quindi di non richiedere circuiteria addizionale per le comunicazioni. Presenta però diversi svantaggi, che in alcuni casi la rendono addirittura impraticabile: la CPU è praticamente occupata per la maggior parte del tempo a interrogare dispositivi, mentre solo una minima parte di questo è impiegata per la comunicazione vera e propria. Inoltre il tempo di risposta della CPU alle richieste dei dispositivi può essere in certi casi troppo lungo: se ad esempio i dispositivi sono parecchi, e uno di essi richiede un'operazione di I/O immediatamente dopo essere stato interrogato, deve aspettare parecchio prima di essere servito; in certi casi questo può comportare la perdita di dati.
La seconda tecnica, che elimina questi svantaggi, è quella degli interrupt.
Sono gli stessi dispositivi a segnalare alla CPU la richiesta di un'operazione di I/O, inviando un segnale, (chiamato appunto interrupt, o interruzione) per richiedere alla CPU di interrompere il programma che sta eseguendo, per effettuare l'operazione di I/O. Il servizio di interrupt avviene con un salto a una routine, che prende il nome di routine di servizio dell'interrupt, la quale provvede ad eseguire l'operazione richiesta.
Questa tecnica presenta molte analogie con l'esecuzione della subroutine, con la differenza che l'esecuzione della routine di servizio avviene in momenti non prevedibili dal programma, che dipendono dalle esigenze dei dispositivi esterni.

È chiaro che la tecnica degli interrupt elimina gli svantaggi cui si era prima accennato. Il tempo di risposta è infatti limitato solo dalla *velocità della CPU* per trasferire il controllo da una zona a un'altra della memoria; inoltre l'unità centrale può dedicarsi ad altri programmi, utilizzando in modo più efficiente il suo tempo.

La differenza tra polling e interrupt è la stessa che ci sarebbe tra aprire la porta di casa ad intervalli di tempo prefissati, per vedere se *c'è qualcuno, e il* rispondere al trillo del campanello. La perdita di tempo nel primo caso è evidente, come lo è la scomodità del servizio per chi, desiderando comunicare, deve attendere il successivo controllo per poterlo fare. Dobbiamo però dire che per la gestione delle interruzioni, oltre al dispositivo (il campanello) che segnali la richiesa di comunicazione è *necessario predisporre* un hardware più complicato. Prima di tutto occorrono una o più linee della CPU dedicate al ricevimento dei segnali di interrupt; in più occorre che lo stesso interrupt si faccia riconoscere dalla CPU, e anche questo rende più complessi i collegamenti; infine, anche se non sempre, è necessaria della circuiteria che consenta il servizio dei diversi dispositivi, in *base alla priorità* prefissata.
Tuttavia, dal momento che la tecnica dell'interrupt è così vantaggiosa dal punto di vista pratico, quasi tutti i costruttori di computer - escludendo casi particolari - vi ricorrono abitualmente. La *possibilità di utilizzo* degli interrupt è inoltre estesa anche ai normali programmatori da particolari istruzioni di cui è dotata la CPU. È quindi importante sapere dell'esistenza degli interrupt per due motivi: 1) chiunque li può usare nei propri programmi assembler; 2) il loro funzionamento aiuta a capire tutte quelle azioni che il computer esegue senza che vi sia un intervento diretto della *mano dell'uomo.*

LINGUAGGIO

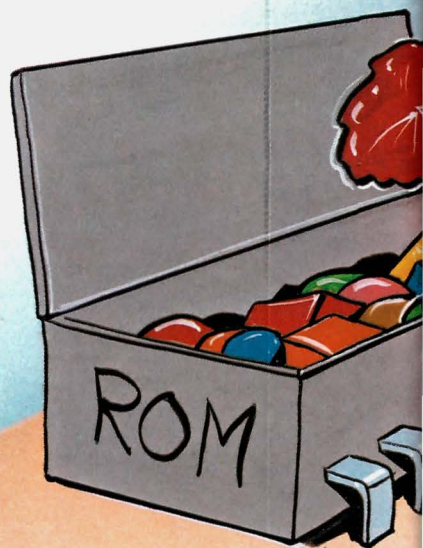
Usare la ROM

Esistono molte routine del sistema operativo che possono consentire al programmatore esperto di risparmiare il tempo e la fatica di doverle riscrivere. I progettisti di calcolatori strutturano infatti le routine di sistema in maniera tale da poterle considerare come dei normali sottoprogrammi, richiamabili in qualsiasi momento sia da BASIC che da Assembler. Il vantaggio di una simile soluzione è più che evidente: si evita ai programmatori di affrontare tutte le volte gli stessi problemi (per esempio la visualizzazione dei risultati), consentendo loro di concentrarsi sul problema specifico piuttosto che sul problema generale. Inoltre le routine di sistema - poste nella memoria ROM e quindi non cancellabili - sono scritte e controllate da programmatori professionisti, con la conseguente garanzia di

sicuro funzionamento. Per poter utilizzare una qualsiasi di queste routine l'unica cosa che si deve conoscere è l'indirizzo di partenza, oltre naturalmente ai registri del microprocessore che vengono interessati dalla routine stessa. Esistono diversi manuali che descrivono con notevole precisione tutte queste routine; noi tratteremo le principali, spiegando quindi (ed è questo che conta) come utilizzarle nei programmi.
Ciascuna di queste routine possiede un particolare nome mnemonico - assegnato dalla casa madre - che le permette di essere distinta in modo semplice e immediato dalle altre. La seguente tabella contiene quindi, oltre all'indirizzo di partenza, anche il nome di ciascuna routine.

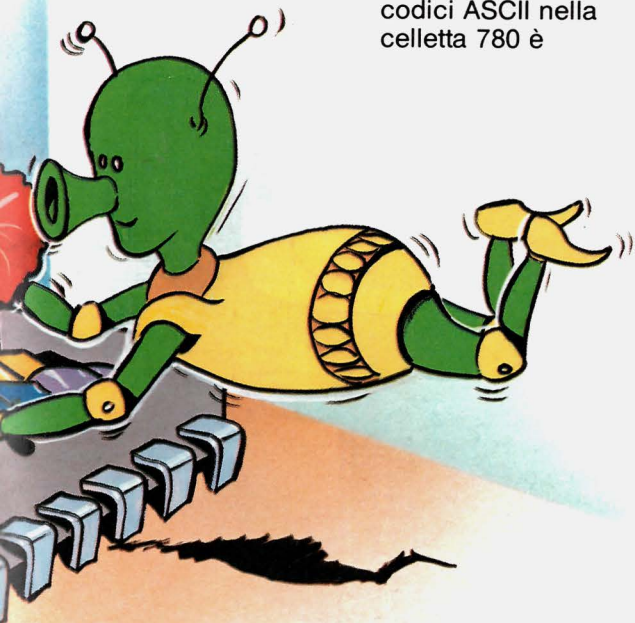
NOME	INDIRIZZO	SCOPO
ACPTR	65445	input porta seriale
CHKIN	65478	OPEN canale seriale
CHKOUT	65481	OPEN canale output
CHRIN	65487	input canale
CHROUT	65490	output canale
CIOUT	65448	output porta seriale
CLALL	65511	CLOSE canali e file
CLOSE	65475	CLOSE file logico
CLRCHN	65484	CLOSE canali I/O
GETIN	65512	GET car. buff. tast.
IOBASE	65523	ind. base dev. I/O
LOAD	65493	LOAD
MEMBOT	65436	Read/Set inizio mem.
MEMTOP	65433	Read/Set fine mem.
OPEN	65472	OPEN
PLOT	65520	Read/Set pos. curs.
RDTIM	65502	legge il clock
RESTOR	65415	riprist. vett. e I/O
SAVE	65496	SAVE
SCNKEY	65439	scansione tastiera
SETMSG	65424	messaggi KERNAL
STOP	65505	STOP

Come esempio di utilizzo di una di queste routine, vediamo in che modo è possibile stampare qualcosa sullo schermo. Dalla tabella appena scritta si deduce che la routine adibita alla visualizzazione dei caratteri sul video è CHROUT; essa richiede semplicemente che, prima della chiamata, il codice ASCII del carattere da visualizzare venga posto nell'accumulatore. Dovremo quindi memorizzare un certo codice nell'accumulatore e chiamare la routine tante volte quanti saranno i caratteri che vogliamo far stampare. Per stampare la parola "ciao", potremo quindi scrivere:

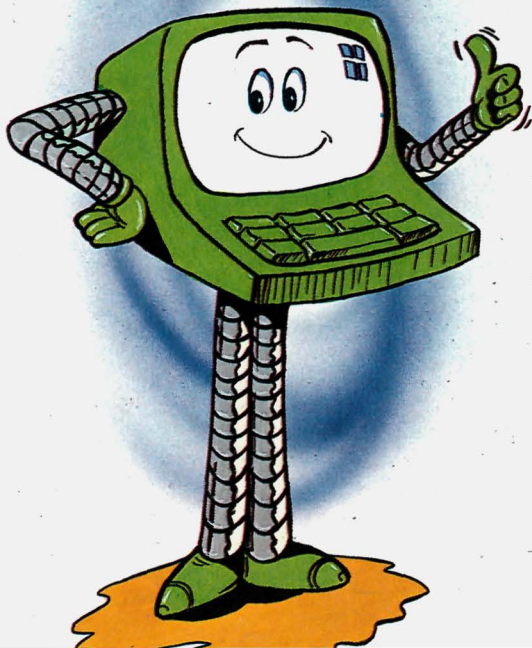
```
LDA #$43    ;ASCII di "C"
JSR $FFD2

LDA #$49    ;ASCII di "I"
JSR $FFD2

LDA #$41    ;ASCII di "A"
JSR $FFD2

LDA #$4F    ;ASCII di "O"
JSR $FFD2
```

Come puoi vedere, il fatto di conoscere l'esistenza di CHROUT ci ha evitato qualsiasi preoccupazione per quanto riguarda l'uscita dei risultati sul video. La chiamata avviene semplicemente attraverso la solita istruzione JSR (Jump to SubRoutine), a ulteriore dimostrazione che nella ROM le routine si

trovano - come è d'altra parte logico che sia - scritte in forma di sottoprogrammi.
Se avessimo voluto risolvere lo stesso problema in BASIC senza dover ricorrere all'istruzione PRINT, avremmo potuto scrivere:

```
10 FOR I=1 TO 4
20 READ A:POKE 780,A
30 SYS(65490)
40 NEXT I
50 DATA 67,73,65,79
```

Il programma è in pratica l'equivalente BASIC di quanto abbiamo appena visto in Assembler. Con un ciclo FOR....NEXT si leggono l'uno dopo l'altro i quattro codici ASCII corrispondenti ai caratteri della parola CIAO, si memorizzano nella locazione 780, visualizzandoli quindi sullo schermo.
La preventiva memorizzazione dei codici ASCII nella celletta 780 è necessaria, perché questa locazione rappresenta uno pseudo-registro, che al momento di una chiamata SYS diventa il valore attuale del registro A (accumulatore) della CPU. In altre parole, al momento di eseguire una SYS (immediatamente prima) l'interprete BASIC trasferisce il valore di questa locazione nel registro A; ai fini del programma la locazione 780 può pertanto essere tranquillamente identificata con l'accumulatore stesso. È chiaro come questo programma non sia assolutamente da utilizzare nell'uso "normale": con una semplicissima PRINT avremmo infatti svolto lo stesso compito di questo ciclo FOR....NEXT, con un minor utilizzo della memoria e oltretutto a vantaggio della velocità di esecuzione. Lo scopo dell'esempio era esclusivamente quello di farti vedere come è possibile utilizzare la ROM sia nei programmi BASIC che in quelli Assembler, adottando la tecnica del ricorso ai sottoprogrammi di sistema del tuo computer.

VIC musicista

A titolo di applicazione di quanto abbiamo visto sulla gestione degli interrupt, proviamo ora ad esaminare un breve programma, nel quale - utilizzando la routine di interrupt - faremo suonare al VIC una famosa musichetta, mentre svolge altri programmi.
La routine in Assembler che segue inizializza i puntatori in 788 e 789 (che servono per indicare - in caso di interrupt - a quale locazione si trova il programma da eseguire) con il valore 7568, che è l'indirizzo di partenza del programma, settando inoltre con 8 bit il byte 175 (usato nella routine musicale come FLAG).

INDIRIZZO	ISTRUZIONI	SIGNIFICATO
7552	SEI	disabilita gli interrupt
7553	LDA#$90	carica in A il valore 144 ($90)
7555	STA$0314	memorizza il numero nel byte 788 ($314)
7558	LDA#$1D	carica in A il valore 29 ($1D)
7560	STA#$0315	memorizza il numero nel byte 789 ($315)
7563	LDA#$FF	carica in A il valore 255 ($FF)
7565	STA$AF	memorizza il numero nel byte 175 ($AF)
7567	RTS	esce dal sottoprogramma

Qui sotto puoi invece vedere la routine musicale vera e propria:

INDIRIZZO	ISTRUZIONI	SIGNIFICATO
7568	LDA AF	carica il flag 175 ($AF) nell'accumulatore
7570	BEQ 1DAE	salta, se zero, a 7598 ($1DAE)
7572	DEC B0	decrementa il contenuto del byte 176 ($B0)
7574	BNE 1DAE	se non è zero, salta a 7598 ($1DAE)
7576	LDA #$13	carica in accumulatore il numero 19 ($13)
7578	STA B0	memorizza il numero nel byte 176 ($B0)
7580	LDY B1	carica nel reg. Y il contenuto del byte 177 ($B1)
7582	INC B1	aggiunge 1 in 177 ($B1)
7584	CPY #$30	confronta Y con 48
7586	BNE 1DA8	se non è verificato, salta a 7592 ($1DA8)
7588	LDA #$01	carica in A il valore 1
7590	STA B1	memorizza il numero nel byte 177 ($B1)
7592	LDA 1DCF,Y	carica in A il contenuto di $1DCF+Y
7595	STA 900A	memorizza la nota nel byte 36874 ($900A)
7598	JMP EABF	salta all'indirizzo 60095 ($EABF) della normale routine di interrupt

Il byte 176 ($B0) fa da temporizzatore alla nota, che dura 19 chiamate di interrupt (circa 0,3 secondi). Il byte 177 ($B1) contiene (con l'aggiunta del registro Y) il puntatore alla nota corrente; le note sono 48 e vengono poste nel generatore di suono, byte 36874 ($900A), e il programma BASIC pone poi il volume nel byte 36878.

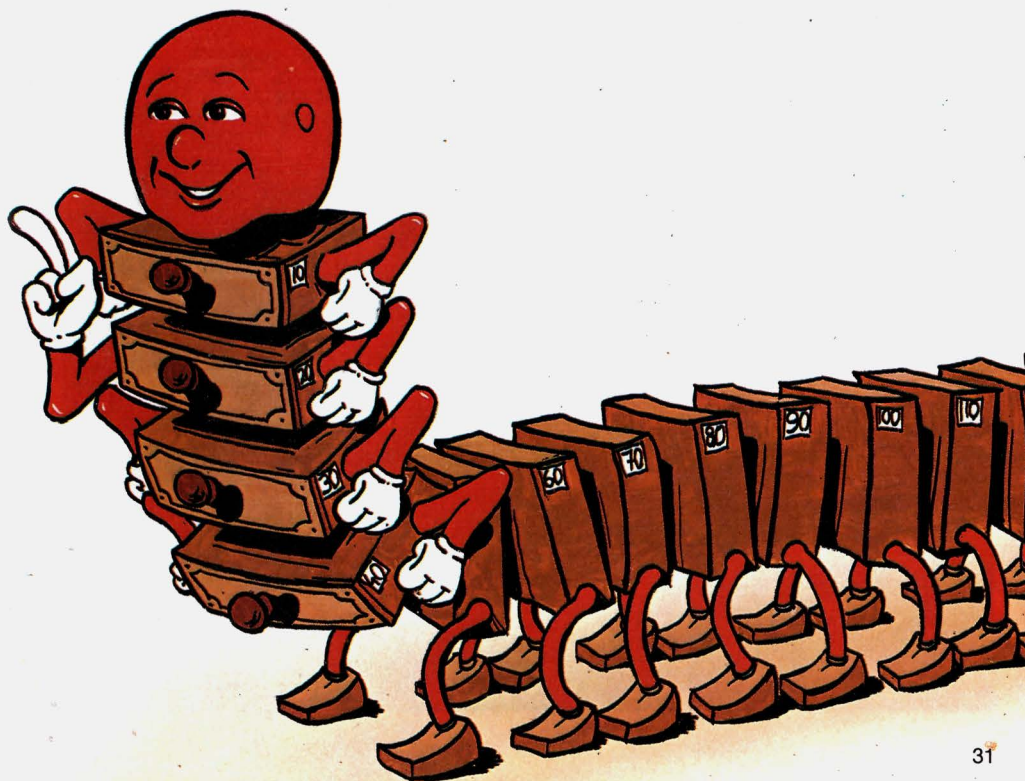
Il listato BASIC che carica ed esegue la nostra routine Assembler è questo:

```
 20 FORI=0TO48:READA:POKE7552+I,A:NEXT
 30 FORI=0TO47:READA:POKE7632+I,A:NEXT
 40 POKE36878,15:POKE176,19:POKE177,1
 45 POKE643,128:POKE644,29:SYS7552:END
 99 REM *** DATI PROGRAMMA ***
100 DATA 120,169,144,141,20,3,169,29,141,21
110 DATA 3,169,255,133,175,96,165,175,240,26
120 DATA 198,176,208,22,169,19,133,176,164
130 DATA 177,230,177,192,48,208,4,169,1,133
140 DATA 177,185,207,29,141,10,144,76,191,234
150 REM ***** DATI MUSICA *****
160 DATA 207,217,223,227,228,227,223,217,207
170 DATA 217,223,227,228,227,223,217,183,199
180 DATA 207,212,215,212,207,199,207,217,223
190 DATA 227,228,227,223,217,191,203,212,203
200 DATA 183,199,207,199,207,217,219,221,223
210 DATA 191,223,191
```

Nella linea 20 vengono caricati, a partire dall'indirizzo 7552, i 49 byte delle due routine in linguaggio macchina.
Nella linea 30 vengono caricate le 48 note a partire dal byte 7632.
Nella linea 40 viene posto a 15 il byte 36878 del volume, e vengono inizializzati i due contatori usati dal programma in linguaggio macchina; il byte 176 ($B0) con il numero 19 per la durata della nota e il byte 177 ($B1) con il numero 1.
Nella linea 45 con le istruzioni POKE vengono modificati i contenuti dei due byte 643 e 644, che danno l'indirizzo di fine memoria; essi vengono posti all'indirizzo 7552 (29*256+128=7552=$ 1D80). In tal modo il programma in linguaggio macchina che si trova a quell'indirizzo non viene disturbato. Inoltre, con SYS 7552 viene lanciato il programma Assembler che parte appunto da quell'indirizzo.
Una volta eseguito il programma potrai usare il BASIC normalmente, mentre sentirai il VIC suonare una musichetta: ogni sessantesimo di secondo, grazie alla modifica dell'interrupt, verrà infatti attivata la routine in linguaggio macchina.

Animazione in LM

Muovere sul video uno sprite in LM è tutt'altra cosa rispetto al BASIC. Il cuore del programma è nel codice macchina pokato a partire dall'indirizzo 673.

```
 10 DATA 0, 16, 144, 87, 61, 253, 12, 12, 56, 126, 254, 215, 255, 171, 130, 198, 13, 13, 15,
    15, 15
 15 DATA 15, 63, 63, 252, 254, 255, 255, 255, 255, 255, 255, 63, 127, 255, 255, 127, 61,
    120, 240
 20 DATA 255, 254, 252, 252, 254, 255, 247, 98
 50 DATA 160, 0, 152, 160, 20, 162, 255, 202, 208, 253, 136, 208, 248, 168
 55 DATA 169, 32, 153, 0, 30, 153, 1, 30, 153, 22, 30, 153, 23, 30, 153, 44, 30, 153, 45, 30
 60 DATA 166, 203, 224, 41, 208, 1, 96, 224, 33, 208, 5, 192, 0, 240, 10, 136
 65 DATA 224, 30, 208, 5, 192, 20, 240, 1, 200
 70 DATA 24, 169, 0, 153, 0, 30, 105, 1, 153, 1, 30, 105, 1, 153, 22, 30, 105, 1
 75 DATA 153, 23, 30, 105, 1, 153, 44, 30, 105, 1, 153, 45, 30, 76, 163, 2
100 POKE 52, 28 : POKE 56, 28
105 FOR C = 7168 TO 7679 : POKE C, 0 : NEXT
110 FOR C = 7168 TO 7215 : READ A : POKE C, A : NEXT
115 FOR C = 673 TO 765 : READ A : POKE C, A : NEXT
120 PRINT "♥" : POKE 36869, 255 : POKE 36879, 1
125 SYS 673 : POKE 36869, 240
130 END
READY.
```

Disassemblato del LM

Ecco il disassemblato del programma in LM.

```
         LDY,  0
START    TYA                       ; SALVA Y NELL'ACCUMULATORE
         LDY   #14                 ; CICLO DI RITARDO
LOOP1    LDX   #FF
LOOP2    DEX
         BNE   LOOP2
         DEY
         BNE   LOOP1
         TAY                       ; RECUPERA Y
         LDA   #20
         STA   VIDEO, Y            ; CANCELLA SPRITE
         STA   VIDEO + 1, Y
         STA   VIDEO + 14, Y
         STA   VIDEO + 15, Y
         STA   VIDEO + 2C, Y
         STA   VIDEO + 2D, Y
         LDX   TAST                ; TASTO PREMUTO
         CMX   #29                 ; TASTO = "S"
         BNE   CONF1               ; NO, VAI A CONF1
         RTS                       ; SI, RETURN
CONF1    CMX   #21                 ; TASTO = "Z"
         BNE   CONF2               ; NO, VAI A CONF2
         CMY   #0                  ; RAGGIUNTO MARGINE SINISTRO
         BEQ   DISPL               ; SI, VAI A DISPL
         DEY                       ; NO, DECREMENTA Y
CONF2    CMX   #1E                 ; TASTO = "/"
         BNE   DISPL               ; NO, VAI A DISPL
         CMY   #14                 ; RAGGIUNTO MARGINE DESTRO
         BEQ   DISPL               ; SI, VAI A DISPL
         INY                       ; INCREMENTA Y
DISPL    CLC
         LDA   #0                  ; VISUALIZZA SPRITE
         STA   VIDEO, Y
         ADC   #1
         STA   VIDEO + 1, Y
         ADC   #1
         STA   VIDEO + 14, Y
         ADC   #1
         STA   VIDEO + 15, Y
         ADC   #1
         STA   VIDEO + 2C, Y
         ADC   #1
         STA   VIDEO + 2D, Y
         JMP   START               ; RIESEGUI IL PROGRAMMA

SYMBOL VALUE

START  02A3    LOOP1  02A6    LOOP2  02A8    VIDEO  1E00
CONF1  02CA    CONF2  02D3    DISPL  02DC    TAST   00CB
```